

;login:

The USENIX Association Newsletter

Volume 8 Number 6

December 1983

CONTENTS

The Evolution of the Berkeley UNIX Project	3
Results of the Voting on the Bylaws Revision	4
USENIX Conference Proceedings Available	4
Schedule of USENIX Technical Sessions at UniForum	5
Wednesday, January 18	5
1:30— 3:00 Networking	5
3:30— 5:00 Distributed UNIX	5
7:00— 8:30 C Language Evolution & Standardization	5
Thursday, January 19	5
8:30—10:00 Compilers and Languages	5
10:30—12:00 UNIX Directions	6
1:30— 3:00 Applications	6
3:30— 5:00 Implementations I	6
5:00— 7:00 Meeting of the USENIX Board with the Members	6
Friday, January 20	7
8:30—10:00 Implementations II	7
10:30—12:00 Communications	7
1:30— 3:00 Graphics	7
3:30— 5:00 Real-Time UNIX	7
Abstracts for USENIX Technical Sessions at UniForum	9
Networking	9
Distributed UNIX	12
Languages and Compilers	15
UNIX Directions	17
Applications	19
Implementations I	22
Implementations II	26
Communications	29
Graphics	32
Real-Time UNIX	34

NOTICE

;login: is the official newsletter of the USENIX Association, and is sent free of charge to Individual, Public and Institutional members of the Association.

The USENIX Association is an organization of AT&T licensees, sub-licensees, and other persons formed for the purpose of exchanging information and ideas about UNIX* and UNIX-like operating systems and the C programming language. It is a not-for-profit corporation incorporated under the laws of the State of Delaware. The officers of the Association are:

President	Lou Katz	Directors	Bruce S. Borden
Vice-President	John L. Donnelly		Alan G. Nemeth
Secretary	Lewis Law		Deborah K. Scherrer
Treasurer	Thomas Ferrin		Waldo M. Wedel

The editor of *;login:* is Tom Strong.

Member services are provided through the Association office. Membership information can be obtained from the office:

USENIX Association
P.O. Box 7
El Cerrito, CA 94530
(415) 528-UNIX

Members of the UNIX community are heartily encouraged to contribute articles and suggestions for *;login:*. Your contributions may be sent to the editor electronically at

ucbvax!g:usenix

or through the U.S. mail to the Association office at the address above. The USENIX Association reserves the right to edit submitted material.

;login: is produced on UNIX using *troff* and a variation of the *—me* macros. We appreciate receiving your contributions in *n/troff* input format, using any macro package. If you contribute hardcopy articles please leave left and right margins of 1" and a top margin of 1½" and a bottom margin of 1¼". Hardcopy output from a line printer or most dot-matrix printers is not reproducible.

This newsletter may contain information covered by one or more licenses, copyrights, and/or non-disclosure agreements. Permission to copy without fee all or part of this newsletter is granted only to Institutional members of the USENIX Association, provided that copies are made only for internal use at the member campus or plant site. No other reproduction of this newsletter in its entirety or in part may be made without written permission of the Association.

*UNIX is a trademark of Bell Laboratories.

The Evolution of the Berkeley UNIX Project

[Received from the Computer Systems Research Group, U.C. Berkeley]

The distribution of Berkeley UNIX 4.2BSD to licensed installations began on September 30, 1983, and is proceeding quite smoothly. In the meantime, the Computer Systems Research Group of the University of California at Berkeley (CSRG) has started working on four new research projects, in addition to further tuning and refining the facilities of 4.2BSD. As of August 1, CSRG has been headed by Mike Karels, who previously worked with the Berkeley PDP-11 distribution. He is working under the guidance of Prof. Domenico Ferrari while Prof. Robert S. Fabry is on sabbatical this year. Plans have been prepared for the next three years in each of the four areas. The projects will be supported by a new three-year contract from the Defense Advanced Research Projects Agency.

The first project will study various issues related to the design of mechanisms for distributed access of files and other resources in a network of single-user workstations running under Berkeley UNIX. The issues include the performance effects of the amount and use of local storage present in each workstation, the design of flow control policies to prevent saturation of such remote facilities as file servers, and the tradeoffs between autonomy and transparency in accessing distributed objects.

Determining the cost-performance impact of several decisions to be made when designing a distributed name server is the main goal of the second project. Various lookup algorithms, local caching, and cache validation policies will be investigated. A Berkeley UNIX-based Name Domain server for the DARPA Internet is being developed to run experiments and to provide parameters for the modeling part of the study.

The third project is concerned with evaluating various metrics and policies for automatic load balancing in distributed Berkeley UNIX systems. All the configurations being considered are based on a local-area network, and include single-user workstations as well as multiple-user interactive machines. The goal of this effort is to design and implement a viable load balancing scheme that can be tuned to the characteristics of different environments and host configurations.

The researchers involved in the fourth project are designing a distributed measurement instrument and a distributed program debugger. The two problems are being attacked together because of their common need for remote process control functions. The measurement instrument should, among other capabilities, allow the experimenters to create and sustain an artificial load on a distributed system, to control remote software monitors, to capture inter-process communications, and to keep the clocks of the various hosts on a local-area network in as full synchronization as possible. The debugger should allow programmers to control the state of their distributed applications, and verify the correctness of their assumptions about synchronization and event ordering.

The addition to 4.2BSD of sub-network routing, allowing logical and physical networks (or external and internal addressing) to be different, and the revision of the terminal line disciplines to make them more consistent with the network interface, thereby improving remote terminal facilities, are, among the other projects being considered, the most likely to be undertaken in the near future.

At this time, there are no specific plans for future releases of 4BSD; as the system evolves, the additions will be incorporated into a new distribution when appropriate.

;login:

Results of the Voting on the Bylaws Revision

The new bylaws have been accepted, with the following vote:

For revisions	158
Against revisions	4
Invalid ballots	13

The vote meets the requirements of a 2/3 majority for acceptance and the accepting majority representing at least 1/3 of the total voting membership (411).

Deborah K. Scherrer, Director

USENIX Conference Proceedings Available

San Diego Conference

Copies of the proceedings of the San Diego UNICOM conference are still available from the Software Tools Users Group. They are over 350 pages long and include all papers presented by the speakers as well as reports on many of the presentations.

The price is \$25 per copy, plus \$10 per copy for overseas postage. Send your check or money order made out to "Software Tools Users Group" to:

STUG
1259 El Camino Real, #242
Menlo Park, CA 94025

Toronto Conference

Copies of the Toronto Proceedings are available for purchase from the USENIX office. The price is \$30 per copy, plus \$15 per copy for overseas postage. Payment **must** accompany your order.

Schedule of USENIX Technical Sessions at UniForum

Wednesday, January 18

1:30— 3:00 **Networks — Aspects of Networking under UNIX**

Chair: Thomas Ferrin, University of California at San Francisco

1:30— 1:55 Driver-Based Protocol Implementations

Greg Chesson, Silicon Graphics

1:55— 2:20 The Excelan TCP/IP Protocol Package

Bruce Borden, Silicon Graphics, Inc.

Bill Northlich, Northlich Computer Systems & Software, Inc.

2:20— 2:40 Worknet: A Xenix-Based Merged Filesystem Network

Alan Greenspan, Altos Computer Systems

2:40— 3:00 CSNET Grows Up

Michael T. O'Brien, The Rand Corporation

Daniel B. Long, Bolt Beranek and Newman, Inc.

3:00— 3:30 BREAK

3:30— 5:00 **Distributed UNIX — Multiprocessing under UNIX**

Chair: Alan Nemeth, Prime Computer Inc.

3:30— 3:50 Software Administration over Computer Networks — The exptools Experience

Joseph L. Steffan, Bell Laboratories

3:50— 4:10 A Distributed File System for UNIX

Matthew S. Hecht, John R. Levine & Justin Walker,

Interactive Systems Corp.

4:10— 4:30 Multiprocessor Debugging on a Shared Memory System

Chet Britten & Paul Chen, Metheus Corporation

4:30— 5:00 Panel Discussion — Multiprocessing Issues

5:00— 7:00 DINNER

7:00— 8:30 C Language Evolution & Standardization

L. Rosler, Bell Laboratories

Thursday, January 19

8:30—10:00 **Compilers and Languages**

New Languages and Developments in Familiar Languages

Chair: Louis Salkind, New York University

8:30— 9:00 The Evolution of the Portable C Compiler

S. C. Johnson, AT&T Bell Laboratories

9:00— 9:20 How to Feel Better about Knowing It is Written in C

Alan R. Feuer, Catalytix Corporation

9:20— 9:40 An Implementation of The B Programming Language

Lambert Meertens & Steven Pemberton,

Centrum voor Wiskunde en Informatica

9:40—10:00 Object-oriented Programming in C Language

Brad J. Cox, Productivity Products, Inc.

10:00—10:30 BREAK

;login:

10:30—12:00 **Over Under Sideways Down, Backwards Forwards Square & Round
UNIX Directions**

Chair: Brian E. Redman, Central Services Organization
Bell Operating Companies

10:30—10:45 Behind Every Binary License is the UNIX Heritage

B. E. Redman, P. E. Parseghian, Bell Laboratories

10:45—11:00 How did UNIX Get Here?

A Sketchy History of the Politics of UNIX Development

Andrew Tannenbaum, MASSCOMP

11:00—11:15 A Proposed Syntax Standard for UNIX System Commands

Kathleen Hemenway & Helen Armitage, Bell Laboratories

11:15—11:30 Decision Avoidance; UNIX from DEC, Finally

Armando Stettner, Digital Equipment Corporation

11:30—12:00 Panel Discussion

12:00— 1:30 LUNCH

1:30— 3:00 **Applications**

Chair: Reidar Bornholdt, Columbia University

1:30— 1:45 MLE (Multi-Lingual Editor)

Scott Bradner, Harvard University

1:45— 2:05 MPS: A UNIX-Based Microcomputer Message Switching System

T. Scott Pyne & Joseph S. D. Yao, Hadron Inc.

2:05— 2:20 Taming The Beast (An RSX Emulator for UNIX)

Daniel R. Strick, University of Pittsburgh

2:20— 2:40 A UNIX-Based Color Graphics Workstation

Rex McDowell, Metheus Corporation

2:40— 3:00 User-Level Window Managers for UNIX

Robert J. K. Jacob, Naval Research Laboratory

3:00— 3:30 BREAK

3:30— 5:00 **Implementations I**

Chair: Michael O'Dell, Group L

3:30— 3:50 The UNIX Paging System

Keith Kelleman, Bell Laboratories

3:50— 4:05 Providing a Job Control Facility in UNIX System V

W. P. Weber Jr. & L. S. Weisbrot, Bell Laboratories

4:05— 4:20 Loadable Virtual Disk Device Driver and Server

Thomas A. Alborough, Creare R&D

4:20— 4:45 OSx: Towards a Single UNIX System for Superminis

Ross A. Bott, Pyramid Technology Inc.

4:45— 5:00 New ½ Inch Tape Options and Trade-Offs for 4BSD on DEC VAX Processors

Robert J. Kridle, mt xinu, inc.

5:00— 7:00 Meeting of the USENIX Board with the Members

;login:

Friday, January 20

8:30–10:00 **Implementations II**

Chair: Joseph S. D. Yao, Hadron, Inc.

8:30– 8:50 A Layered Implementation of the UNIX Kernel
on the HP9000 Series 500 Computer

Jeff Lindberg, Hewlett-Packard

8:50– 9:05 Porting Xenix to the Unmapped 8086

John Bruno Hare & Dean Thomas, The Santa Cruz Operation, Inc.

9:05– 9:25 Writing Device Drivers for Xenix Systems

Jean McNamara, Paresch Vaish & Richard N. Bryant, Intel Corporation

9:25– 9:45 Transparent Implementation of Shared Libraries

Curtis Downing, Bunker Ramo Information Systems

Frank Farance, Systems Theory Design Corporation

9:45–10:00 UNIX File Systems Optimization on Microcomputer Systems

David Robboy, Intel Corporation

10:00–10:30 BREAK

10:30–12:00 **Communications – UNIX Talks to Itself and Others**

Chair: Mark Horton, Bell Laboratories

10:30–10:45 A UNIX to VAX/VMS Communications Link

Clifford N. Cary, Creare R&D

10:45–11:00 Loving UUCP, or How I Spent My Summer Vacation

P. Honeyman, D. A. Nowitz, B. E. Redman, Bell Laboratories

11:00–11:15 New Implementations of UUCP

D. A. Nowitz, P. Honeyman, B. E. Redman, Bell Laboratories

11:15–11:30 Mapping the UUCP Network

Rob Kolstad & Karen Summers-Horton, Bell Laboratories

11:30–12:00 UUCP-USENET Panel – UUCP/USENET issues

12:00– 1:30 LUNCH

1:30– 3:00 **Graphics Under UNIX**

Chair: Noel Kropf, Columbia University

1:30– 2:00 UNIX as a Development Base for High Performance Graphics Applications

Thomas Ferrin, University of California at San Francisco

2:00– 2:20 DAPS and GSDL: A Procedural Approach to Graphics

Christos Tountas, Graphics Information Systems Technology Inc.

2:20– 2:40 The Design of a Dedicated Graphics Subsystem for a UNIX Machine

Jack Burness, MASSCOMP

2:40– 3:00 Image Processing Work Station

Dale Hensley, TRW

3:00– 3:30 BREAK

3:30– 5:00 **Real-Time Under UNIX**

Chair: Joseph Germann, Sky Computers

3:30– 3:50 Life with UNIX in Real-Time

Steven Polyak, Contel Information Systems

3:50– 4:10 Real-Time Extensions to the UNIX Operating System

Bryon Look & Gary Ho, Hewlett-Packard

4:10– 4:30 UNIX Optimization – UNIX/68000/SKYFFP

Joseph Germann, Sky Computers

4:30– 4:45 Integrating a Peripheral Floating-Point Processor into UNIX

Eryk Vershen, UniSoft Systems

;login:

This page was unintentionally left blank

;login:

Abstracts for USENIX Technical Sessions at UniForum

Networking

1:30–3:00 Wednesday

Driver-Based Protocol Implementations

Greg Chesson

Silicon Graphics
630 Clyde Court
Mountain View, CA 94043
(415) 960-1980

Network implementations under UNIX tend to add layers, primitives, system calls and other software gadgets to the operating system. The result is seldom, possibly never, pleasing. Consider the problem of providing incoming virtual terminal terminations in the operating system. A common technique involves the use of pseudo-typewriters, extra processes, and lots of data movement with no particular performance or complexity advantages. Dennis Ritchie's elegant implementation of kernel coroutines has good, but not spectacular performance, and some complexity. It compares favorably with other approaches, especially pseudo-typewriters.

However, one might argue that all of these software techniques will soon be rendered obsolete by new-generation communications interfaces that provide on-board transport protocols. If we postulate a network device that is well suited to both character and block i/o by virtue of its firmware, the corresponding device driver might look like a combination of a dz/dh and a simple disk driver, e.g. the ancient rf. The main attraction here is that the network would no longer be an alien creation, but would more closely resemble the i/o model that the system prefers.

Since these network devices are not quite yet available, it is interesting as well as useful to investigate software techniques that work with existing devices. The approach used at Silicon Graphics consists of adding a protocol module to an otherwise raw device driver. The resulting driver looks like a character device to the typewriter portion of the system and looks like a block device to block portion of the system. The system "sees" reliable data streams and is not aware that a protocol implementation is lurking in the driver. This means that virtual terminals hook into the operating system at the same place as local terminals and have identical modes and character-processing semantics. The block device interface is equally well situated for representing remote files.

An implementation of Xerox's Sequenced Packet Protocol has been tested in various operating systems: Version 7, System V, 4.1, 4.2, and VMS. In each of these cases the protocol part changes very little. The real differences are in the device driver interfaces.

The presentation will focus on the architecture and implementation, and as time permits tuning and debugging issues.

;login:

The Excelan TCP/IP Protocol Package

Bruce Borden

Silicon Graphics, Inc.
630 Clyde Court
Mountain View, CA 94043
(415) 960-1980

Bill Northlich

Northlich Computer System & Software
21 Newell Court
Walnut Creek, CA 94595
(415) 938-9713

The Berkeley/BBN IP/TCP internetwork protocol code was ported to an intelligent front-end Ethernet board. The EXOS/101 Ethernet Front-End Processor from Excelan is a Multibus DMA board with an 8MHz 8088 and 128KBytes of ram memory. By porting the protocol code onto the front-end, the host overhead was reduced drastically, and the kernel-resident protocol software was reduced to a small device driver. The implementation mimics the Berkeley socket interface utilizing pseudo-devices and ioctl calls, thus allowing most of the existing network user code to run without modification.

The prom-resident operating system on the EXOS/101 board provides support for multiple processes and communication via message queues. The initial implementation maintains one process per TCP connection, which had to be rewritten to properly handle multiple forks reading and writing the same connection. The current implementation maintains one process which knows how to save its state when "sleep" is called, and to resume a blocked incarnation when "wakeup" is issued. Both of these approaches were possible without any changes to the Berkeley code, only to the interface and support routines.

The host to front-end protocol was designed to place most of the burden on the front-end, and to be completely host operating system independent.

This talk will describe the port to the front-end, and will examine the desirability of this approach for future network protocol implementations.

Worknet: A Xenix based merged filesystem network

Alan Greenspan

Altos Computer Systems
2641 Orchard Park Way
San Jose, CA 95134
(408) 946 - 6700
ucbvax!microso!altos86!alan

Worknet is a merged filesystem network of Altos microcomputers running Xenix. All machines on the network are logically connected by a "super-root" directory which is used to generate pathnames for accessing files network wide. This approach allows the sharing of disk resources and peripherals such as tape drives and printers while maintaining a user interface which is essentially transparent. Running processes on remote CPUs is supported and includes the ability to pipe together processes running on different machines. Support for diskless workstations is provided by including a network pseudo-disk

;login:

driver for pipes and swapping. Protections are extended across the network at login time by verifying passwords and special considerations are given to "setuid" programs.

The implementation involves changes only to the Xenix kernel and the addition of user mode files server processes. This allows applications to access the network without re-compilation or re-linking. At the transport level both Ethernet and RS-422 are supported with protocols compatible with the ISO proposed standard.

CSNET Grows Up

Michael T. O'Brien
The Rand Corporation
1700 Main St.
Santa Monica, CA 90406
Net: obrien@rand-unix

Daniel B. Long
Bolt Beranek and Newman Inc.
10 Moulton St.
Cambridge, MA 02238
Net: long@bbn-unix

The Computer Science Research network (CSNET) is undergoing a transition from a development phase to a member-supported utility. In this paper, we describe the many changes that will result from this transition. Work is proceeding in several areas. Among these are:

New Network Software. Several CSNET members are in the process of becoming CSNET/Telenet sites using the CSNET-developed IP-to-X.25 interface. This software/hardware facility allows a site to connect to the Telenet public packet-switch network, and to use it to communicate with other hosts on both Arpanet and Telenet using the DOD IP/TCP protocols encapsulated in X.25 packets. This development gives member sites who use it the ability to treat Telenet as a type of public Arpanet.

New Mail Software. CSNET is committed to tracking Berkeley UNIX release 4.2BSD and the BBN TCP/IP implementation. Transporting and testing the software will be completed in early 1984. Phonenet software will be available for V7, 4.1BSD, 4.2BSD, and (using member-contributed software) VMS and other systems with Pascal. CSNET/Telenet and Nameserver software will be available for 4.2BSD only. A major rewrite of MMDF, the CSNET standard mail transport system, fully supports address standard RFC 822 and has operational and performance improvements in several other areas. In addition, member-contributed facilities exist for connecting MMDF to UUCP and managing mailing lists.

Mail Gateways. Negotiations are under way to connect CSNET with several European and domestic networks. There are already several affiliate members in Canada in the process of connecting to CSNET. An organization in Sweden is soon to become a gateway to the Swedish University Network using the CSNET/Telenet facility. In addition, a CSNET/BITNET gateway will be operational by the end of 1983.

;login:

Distributed UNIX

3:30—5:00 Wednesday

Software Administration over Computer Networks The Exptools Experience

Joseph L. Steffen

Bell Laboratories

Naperville, Illinois 60566

(312) 979-5381

harpo!ihnp4!ihulh!steffen

This abstract describes a working method for large-scale, distributed administration of software over computer networks. It is being used for the experimental tool package (exptools) at Bell Labs, which is a collection of the latest programming and text processing tools supported by the authors or other individuals rather than an official organization. It contains more than 60 tools maintained by over 20 people on more than 100 machines of four types connected by two different networks. There is an overall coordinator who controls the installation of new tools to insure a uniform tool set across all machines; but the individual tool supporters actually maintain the tools, that is, fix bugs and install new releases.

The novel features of *exptools* administration are:

- The division of software administration between overall coordination and individual tool support.

- The use of existing computer network capabilities (file transfer and remote command execution) to install and update tools.

The latter avoids manually logging into each machine, which has several benefits:

- A tool can be updated on all machines in hours instead of days or weeks.

- Adding more machines does not significantly increase administrative effort.

- Tools can be supported by people or organizations outside the computer center because the coordinator has final control.

- Work done by the coordinator is minimized, which allows one person to administer scores of machines.

- Tool administration can be separated from machine administration, i.e., the local system administrator does not install or update tools.

;login:

A Distributed File System for UNIX

Matthew S. Hecht
John R. Levine
Justin C. Walker

INTERACTIVE Systems Corporation
8689 Grovemont Circle
Gaithersburg, Maryland 20877
(202) 789-1155
allegro!ima!matthew

and

441 Stuart Street
Boston, Massachusetts 02116

We describe a design for achieving user-level transparent access to remote files in a local area network of UNIX systems. This design features (1) a *remote mount* system call that allows the mounting of a remote directory, and (2) a specialized *remote function call* mechanism that allows one UNIX kernel to call functions in another. The talk focuses on design problems that arise and the solutions that we chose.

The problem we solve here is how to make access to local and remote UNIX files indistinguishable to users; that is, to users the syntax and semantics of local and remote file access are identical. For example, the user of a command like

```
cp x y
```

where *x* and *y* are arbitrary pathnames, can be oblivious to the location of files *x* and *y* (one or both may be local or remote) since the underlying UNIX system calls do the right things in either case.

Transparent access to remote files in a local area network of UNIX systems provides new opportunities for file sharing. Independent UNIX systems can share files, diskless workstations can obtain file service from another UNIX system, and workstations with low capacity disks can share databases. Transparent access also allows files to be relocated without breaking programs.

Our solution features a remote mount system call, and makes a cut at the i-node function interface. This work contains a novel mix of implementation ideas, yielding a simple, clean, and practical solution. Instead of assigning a remote file-server process to a local user process, we use a pool of kernel file-server processes that feed from a common request queue. In addition, we use a datagram-based, specialized remote function call mechanism that draws ideas from work by Nelson [3].

Related extant work on distributed file systems for UNIX is extensive and growing; we comment on a few related papers here. Our work is similar to that of Luderer and others [2] at Bell Labs on S-UNIX/F-UNIX and to a design of Plexus Computers described by Groff [1]. However, these papers indicate a different process architecture with communication based on virtual circuits. The LOCUS project of Popek and others [4, 6] at UCLA is more ambitious; we do not consider replicated files nor transactions. The COCANET project of Rowe and Birman [5] at UC Berkeley is also more ambitious than our work; we do not handle remote processes.

1. Groff, J.R., "Modified UNIX System Times Network Architecture," *Electronics*, pp. 159-163 (September 22, 1983).
2. Luderer, G.W.R., *et al.*, "A Distributed UNIX System based on a Virtual Circuit Switch," *Proc. 8th Symposium on Operating Systems Principles*, Pacific Grove, Calif., pp. 160-168 (December 1981).
3. Nelson, B.J., "Remote Procedure Call," report number CSL-81-9, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, Calif. (May 1981).

;login:

4. Popek, G., *et al.*, *Proc. 8th Symposium on Operating Systems Principles*, Pacific Grove, Calif., pp. 169-177 (December 1981).
5. Rowe, L.A., and Birman, K.P., "A Local Network Based on the UNIX Operating System," *IEEE Trans. on Software Engr.*, Vol. SE-8, No. 2, pp. 137-146 (March 1982).
6. Walker, B., *et al.*, "The LOCUS Distributed Operating System," *Proc. 9th Symposium on Operating Systems Principles*, Bretton Woods, New Hampshire, pp. 49-70 (October 1983).

Multiprocessor Debugging On A Shared Memory System

Chet Britten

Paul Chen

Metheus Corporation

5289 NE Elam Young Parkway

Hillsboro, Oregon 97123

decvax!teklabs!ogcvax!metheus!chet

This paper discusses a simple debugging technique used in developing a shared memory multiprocessor 68000 UNIX operating system. Techniques applicable to multiprocessor debugging in general, and those dependent on specific hardware are presented. Some of the problems and trade offs are also covered.

In developing the operating system for the Metheus Lambda 750 VLSI design workstation, we first developed several tools for debugging the kernel. Most of the ideas offered could be used by any implementation. Hardware design considerations to make debugging easier are also discussed.

The debugger we used allows us to run either of two processors individually or concurrently. Once the task of allowing breakpoints and tracing is implemented, capabilities can easily be added. The host debugger running on the VAX allows us to see symbolic C stack traces on either processor, set multiple breakpoints on either processor, and examine memory either in different radices or as 68000 instructions.

We have a debug monitor running on one of the processors. This processor saves its own registers on a trace or breakpoint trap, then starts executing the monitor code. The only thing the other processor has to do, on a breakpoint or trace trap, is to dump its registers, indicate it has done so, wait to be told to proceed, then restore its registers and resume execution. While the second processor is waiting the debug monitor can examine or change any of the processor registers, set or remove breakpoints, or set or clear the trace bit.

Having a multiprocessor debugger for the kernel has much more than paid for the effort to implement it.

;login:

Languages and Compilers

8:30—10:00 Thursday

The Evolution of the C Language and the Portable C Compiler

S. C. Johnson and L. Rosler

AT&T Bell Laboratories
Murray Hill, New Jersey

The C language has continued to evolve since the publication of the Kernighan and Ritchie book in 1978 established a de facto standard. Among the enhancements to date are enumeration data types, a *void* type for functions that return no value, long (more than eight-character) identifiers, and expanded semantics for structures and unions.

Current directions of C evolution include: the declaration of the types of function arguments and the coercion of actual parameters; the ability to specify explicit constants (variables or aggregates whose values cannot change during execution, and which can therefore be compiled into program text space or into read-only memory); and assembly-language windows that allow access to C variables by name. The talk will describe how these developments are taking place at the same time that the language is being standardized by ANSI and IEEE committees.

During the same time period, the compiler technology that developed the Portable C Compiler (which was used to provide more than 30 production compilers on different machines) has evolved into a next generation of compilers called PCC2. These offer an easier porting process and improved maintainability, based on the use of a description of the target machine to produce the code-generation templates, which support a wider variety of machine features.

The talk will touch briefly on other experiments based on C, including the extension of C to support abstract data types ("classes") with operator overloading, and a dialect of C that compiles to silicon integrated circuits.

How to Feel Better about Knowing It is Written in C

Alan R. Feuer

Catalytix Corporation
Cambridge, Massachusetts

C is a controversial programming language praised for its versatility and portability, and criticized for its low-level nature. Even though it ignores many of the adages that good languages should follow, it has become the language of choice for more and more organizations.

Today C is used in environments and for applications far different from those for which it was originally designed. While its spread is heartening to those of us who feel that C is a good tool for producing software, there is one significant aspect of C that we should not ignore: C is a dangerous language.

There are many ways in which the unknowing programmer can fall prey to C's pitfalls: by marching off the end of an array, by referencing through a dangling pointer, by passing the wrong kind of arguments to a function, by forgetting a vital pair of parentheses. In none of these instances is the language much help in finding the error.

;login:

But C is not hopelessly perilous. This paper discusses some of what can be done to make C a safer language and introduces an important new tool for developing reliable software in C.

An Implementation of the B Programming Language

Lambert Meertens
Steven Pemberton

Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam

The B programming language has been designed for use in personal computing. B is a fairly conventional language, but for one thing: the design has fully concentrated on ease of learning and use.

B offers all the advantages of strong typing found in languages such as Pascal. Two nice data types of B are the *list* (a bag or collection of items of one type) and the *table* (an array with indexes of any one type and stored values of any one type). B's lists and tables are fully dynamic. The number of types in B is small: 5, and they may easily be combined to implement other types.

Unlike most typed languages, B has no declarations, the types being inferred from context.

Apart from sheer exhaustion of memory, B does not allow limits to be imposed by the implementation. So identifiers may have any length, numbers may have any magnitude, a list may contain any number of items, and so on.

To support top-down programming B has *refinements*, which behave like parameterless light-weight procedures.

Indentation is used to indicate nesting. This obviates **begin ... end**. It also prevents confusion due to contradiction between indentation and program structure.

Global variables are permanent in B and values such as tables may be extended at will. Therefore, there is no need for an extra file concept and the bother of special file handling commands.

For the language to be available for the intended group of users it must be implemented on relatively cheap computers. However, B is not suitable for tiny computers like 8K 8 bit micro-computers. This would have been 'designing for the past'. Systems that have the capacity needed for a smooth B implementation are just now entering the top segment of the personal-computer market.

A pilot implementation of B has been in operation at the CWI since May 1981, running on a VAX and a PDP 11/45. As a pilot, this version was not intended for general release. A new implementation has just been completed. The new version is more portable. It is coded, like its predecessor, in C. The new version is also faster: the source text of programs is parsed before execution, and the data types of B are implemented in an efficient manner. Also, a B-dedicated editor is included.

;login:

Objective-C Object-oriented Programming in C language

Brad J. Cox

Productivity Products Int.
37 High Rock Road
Sandy Hook, Connecticut 06482
(203) 426 1875

Objective-C is a C pre-processor that adds Smalltalk-80 classes, objects, messages, encapsulation and inheritance to the C programmer's toolkit. No C language capabilities are eliminated, and none are changed. Objective-C allows the programmer to choose between conventional C language tools when efficiency and portability are paramount, and object-oriented power-tools when encapsulation and inheritance are needed for reducing code bulk and complexity. Objective-C is an enhancement to C language, not a stripped-down Smalltalk. It is a production tool for professional C programmers who have determined that C language and the programming style that goes with it fits their problem domain.

UNIX Directions

10:30—12:00 Thursday

Behind Every Binary License is the UNIX Heritage

B. E. Redman
P. E. Parseghian
Bell Laboratories

Lately there seems to be much pessimism in some quarters about the future of the UNIX system. Many who have watched its development from the earliest days feel that the system grows only more corrupted and is steadily declining into mediocrity. These issues are addressed in this presentation, with particular attention to motivations, costs, and benefits.

UNIX was originally designed by a talented fraternity with a clear and common vision for a better computing environment. Now the design is controlled by powers with very different goals in mind (e.g., commercial). Although this influence is not directed toward preserving the "purity" of UNIX in some higher sense, it does not necessarily follow that the system is thus "perverted." UNIX has evolved from a simple, elegant model into one that is certainly complex and often appears downright haphazard. It no longer constitutes a statement of smallness, but appears to be suffering a period of unbounded growth. It is generally accepted that the original systems provided a rich environment for a community of sophisticated computer users, as was intended. More recently it seems that UNIX is touted as a computing panacea, and the compromises that have increased its palatability (indeed, popularity) have reduced its effectiveness. Perhaps the most frightening development is the threat that the "total system" (including source code) will no longer be distributed, or will be available only at prohibitively high costs. Is it legitimate to call such a system "UNIX"?

The term "UNIX" has come to represent more than an operating system or computing environment; it represents philosophies about computing. Although the costs seem high and the motivations impure, we must recognize that an important benefit has been realized: the UNIX philosophy has been spread

;login:

among the computing masses and has influenced the direction of computing. The commercialization of UNIX is responsible for this. Other systems may have been just as revolutionary, but will never have a similar impact because they were kept private.

How did UNIX get here? A Sketchy History of the Politics of UNIX Development

Andrew Tannenbaum
MASSCOMP

One way in which UNIX differs from other operating systems is the environment in which it was developed. On one side, UNIX was developed by various groups within Bell Laboratories, in a company which was restricted by consent decree to produce telecommunications products. On the other side, UNIX was leaked to universities, where hackers took it upon themselves to hack UNIX up good.

This paper will discuss the trials and tribulations of UNIX's youth so that folks might better understand why UNIX grew up the way it did.

A Syntax Standard for UNIX Systems Commands

Kathleen Hemenway
AT&T Bell Laboratories, Murray Hill, NJ

Helene Armitage
AT&T Bell Laboratories, Piscataway, NJ

The syntax of UNIX System commands has led some to criticize the system's user interface. The syntax has been criticized for being inconsistent -- there are subtle variations among commands. For example, while some commands allow more than one argument to follow a single delimiter, other commands don't: *ls -l -t* may be abbreviated to *ls -lt*, but *lpstat -d -r* may not be abbreviated to *lpstat -dr*. Similarly, the significance of white space between arguments varies among commands: Some commands require a space before arguments to options, whereas others don't allow space, and in others space is optional. *sort -o file* and *sort -ofile* are synonyms, while *cut -c list* and *cut -clist* are not. These variations frustrate users when they are learning new commands, and the variations cause users to depend on the manual even for frequently used commands.

Inconsistencies aside, the syntax itself has been criticized. Specifically, the use of single letter options has been questioned. Some critics assert that using a single letter takes terseness too far. They argue that because a given letter typically stands for many different words across commands (e.g., the option "-c" stands for column, character, copy, etc.), it is difficult to learn and remember options. Also, the use of delimiters for options has been questioned. While a few commands use "+" to delimit options that add features, most commands use "-" to delimit all options. Since the hyphen is frequently conceptualized as a minus sign, this usage often seems incongruent.

This paper reports a project that addressed these issues. The goal of the project was to identify and evaluate shortcomings in the command syntax and to identify a syntax standard. The standard is to be used in new commands and as a basis for retrofitting the syntax of existing commands. The standard is intended to exert a constructive influence on the evolution of the command set by establishing a

;login:

template toward which the command set should evolve.

Decision Avoidance; UNIX from DEC, Finally

Armando Stettner
Digital Equipment Corp.
Continental Blvd. 11
Merrimack, NH 03054

Applications

1:30—3:00 Thursday

MLE (Multi-Lingual Editor)

Scott Bradner
Harvard University

A multi-lingual text editor is being developed at the Computer Based Laboratory, Harvard University. It contains non-ASCII characters necessary for a number of European languages: umlauts, grave and acute accents, polish 'L' and 'l' among others. Greek, modern and Classical, is available with full diacriticals. Hebrew, Russian, Coptic and Arabic are being developed. Provisions have been made for languages (such as Arabic) with more cases than just upper and lower.

One key will allow the user to reverse the direction in which the cursor moves, and searches are prompted for, so that a keyboardist can input individual Hebrew words in English text with the cursor moving from left to right, but could also type extended text with cursor moving from right to left. The user will be able to define the all the keys of the keyboard according to taste so that typists accustomed to a specific keyboard will not have to change their habits.

MPS: A UNIX-Based Microcomputer Message Switching System

T. Scott Pyne
Joseph S. D. Yao
Hadron, Inc.
6th Floor
1945 Gallows Road
Vienna, VA 22180
(703) 790-1840

A message switching system which operates under the Xenix variant of the UNIX operating system has been developed by a Hadron team including the authors for use in a law enforcement environment. The system interfaces to Motorola mobile (in-vehicle) digital computer terminals via a radio link and to remote IBM mainframe-class systems via emulation of either 3780 or 3270 protocols. The package

;login:

allows personnel in the field to use the mobile terminals to make inquiries (such as wanted person and/or stolen car checks) directly to state- and national-level data bases such as the FBI's National Crime Information Center, and to forward text messages from unit to unit, or between a unit and the station, in situations where interference, ambiguity, or interception are potential problems.

The project is unique in several ways. We believe it to be the first such implementation to be written in a higher-level language, the first under a general-purpose operating system, and the first on an inexpensive microcomputer. Previous packages of this type were written in assembly language, implemented under proprietary real-time operating systems, and hosted on more expensive minicomputer-class machines. This package is of course written, as the proverb goes, "95% in C."

Taming the Beast (An RSX Emulator for UNIX)

Daniel R. Strick

The notion of simulating the program execution environment of foreign operating systems is not new to UNIX. The programs that maintain these exotic environments are usually called "emulators", probably because the term was used with the RSTS operating system to describe run-time systems that supported the execution of programs built to run on the other DEC PDP11 operating systems, RT11 and RSX11. Since UNIX was originally developed for PDP11s and there was a great deal of software written for the DEC operating systems, it was only natural that emulators would be developed to support the execution of this software in the UNIX environment.

Several RT11 emulators were written, but RSTS and RSX were left alone. RSTS was not attractive because most non-BASIC RSTS programs were run in RT11 emulation mode and RT11 was so much easier to emulate. RSX was probably not attractive because it was perceived primarily as a base for real time applications which could not be supported in a UNIX environment, RSX was much more complicated than RT11, the RSX system calls were not completely documented, and the RSX user interface was unusually ugly.

Time has passed since the first emulators were written. RSTS is nearly extinct, RT11 remains basically a single user system, and RSX has emerged as DEC's candidate for a general purpose operating system for the PDP11. The RSX user interface has improved slightly, but the best place to develop RSX software would be on a UNIX system. An RSX emulator for UNIX would make this easier.

This presentation describes some of the problems encountered when developing an RSX emulator for UNIX. These problems are interesting because they arise from significant incompatibilities between RSX and UNIX and therefore reflect basic differences in design philosophies. One way to learn about UNIX is to study what it is not.

;login:

UNIX Based Color Graphics Workstation

Rex Mc Dowell
Metheus Corporation
Hillsboro, OR

This paper will explain the color graphics implementation of the Metheus Lambda 750 workstation, which integrates Berkeley 4.1 UNIX with high performance color graphics. With color windows and multiple font sizes and styles, a user can customize his UNIX working environment, as well as supplying a powerful graphics workstation.

The Lambda hardware simplified the implementation. A new graphics driver was added, and only minor modifications of the tty driver was made. The Lambda Workstation was designed to support our VLSI Cad software, which needed fast, interactive graphics. Our UNIX runs on 2 12MHZ 68000's, one actually running UNIX, the other is an IO processor, with its own real time kernel. The graphics engine contains another 12MHZ 68000, acting as a display list manager and transformation processor. This 68000 feeds a 2901 bit slice, which does the actual drawing. The bit slice also does clipping, and can draw into 8 bit planes at 4 million pixels a second. The graphics display is 1k x 1k pixels, of which 1k x 768 is visible. The system can be configured with 8, 16, or 24 bits planes. With 24 bit planes, you can display up to 16 million colors at once.

Each shell's color window, which is just another tty to UNIX, starts out with 2 bit planes. UNIX only knows the number of characters per line and the number of lines per page. The display processor gets an actual clist pointer and uses it displays text on the screen. It alone knows about the current window size and font.

An application program can request any number of bit planes from the pool of 24. This request is made to the Window Manager which runs under the real time kernel. The bit planes assigned can be any anywhere, and the program does not know or care which planes it is writing. The Window Manager also assures that the top window's program has all of its bit planes visible in front all other tty windows. After a user is assigned bit planes from the pool, he may then change the color map of his bit planes, or draw something, without affecting any other tty window. Since each program has its own bit planes and color map, no one ever needs to redraw, when tty windows are popped. The system supports a C Graphics Subroutine Library, which is a Core Graphics subset, or the user may use the graphics primitives directly. Once a user has obtained his bit planes, he may break the entire screen into any number of small rectangler sections called zones, or windows in graphics terminology. Each of these zones has its own writemask, drawing colors, size and transformations stack, all of which are handled by the display processor.

;login:

User-Level Window Managers for UNIX

Robert J. K. Jacob

Naval Research Laboratory
Washington, D.C. 20375

Wm manages a collection of windows on a display terminal. Each window has its own shell or other interactive program, running in parallel with those in the other windows. This permits a user to conduct several interactions with the system in parallel, each in its own window. The user can move from one window to another, re-position a window, or create or delete a window at any time without losing his or her place in any of the windows. Windows can overlap or completely obscure one another; obscured windows can be "lifted" up and placed on top of the other windows.

This paper describes how such a window manager for UNIX is implemented entirely as a set of user processes, without modifications to the UNIX kernel. It shows how the simple, but well-chosen facilities provided by the original (Version 6) UNIX kernel are sufficient to support wm. In addition, subsequent versions of wm exploit features of the kernel introduced into newer versions of UNIX to provide faster and more sophisticated window operations, still implemented entirely at the user level.

Implementations I

3:30—5:00 Thursday

Paging in the UNIX System

Keith A. Kelleman

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
201-582-3586

Two research derivatives of the UNIX system have supported paging for several years: Reiser 32V, and BSD. Work is under way at AT&T Bell Labs to bring together the features of both of these systems [and others] to form a demand paged kernel for UNIX System V. This talk will discuss three areas of this work: requirements, architecture, and implementation.

The primary requirement of the paging system is that it be upward compatible with its predecessor. This means that old objects must execute unchanged, and that the meaning of system calls should not be changed. For example, fork(2) should be made efficient rather than inventing a new type of call. A second requirement is that the paging system be based on a general machine-independent architecture.

As part of the paging kernel development, a UNIX system memory management architecture is being defined. The architecture is general enough to support both paging and swapping kernels and many different memory management units. The fundamental component of the architecture is the "region." A region is a kernel data structure that represents a potential virtual address space. The basic operations defined for regions are: create/destroy, attach/detach, copy, change size, and load a file. The defined architecture can support potential new UNIX system features such as shared libraries and mapped files.

;login:

Given a memory management architecture, many demand paging techniques exist to help implement it efficiently. For example demand-zero and copy-on-write pages allow for efficient implementation of `brk(2)` and `fork(2)`. Other techniques such as using referenced and modified page bits, demand loading, pre-paging, clustering, region swapping, and "sticky" regions can be used to minimize disk I/O.

Providing a Job Control Facility in UNIX System V

W. P. Weber Jr.

L. S. Weisbrot

AT&T Bell Laboratories

Murray Hill, New Jersey 07974

When users log into UNIX system, they communicate with a shell that reads commands typed at the terminal and arranges for their execution. The shell defines a user interface that is primarily sequential in nature. A user wishing to execute two commands must wait for the first to complete before the second can begin.

Although some parallelism can be achieved via asynchronous command execution (e.g. "&"), the limitation of a single terminal restricts its usefulness. An asynchronously executed command reading from the terminal will compete for input with the shell. Terminal output is multiplexed together with no way for the user to control it. Modifications to the terminal state are seen by both processes.

To overcome these problems, job control provides the ability to divide a terminal into multiple *virtual* terminals. Each of these virtual terminals may be attached to a shell and placed in its own process group; the result is a *layer*. Users create and manage layers. A user can control which layer a command will run in, which layer keyboard input is directed to, and which layer(s) may write to the screen.

This model has been implemented as two cooperating parts: a driver, *sxt*, and a user-level utility, *shl*. *sxt* provides the mapping between multiple virtual terminals and a single real terminal. *shl* provides user control through the creation and manipulation of layers.

The advantages of this model of job control are reflected in its implementation. With the exception of minor changes to the terminal device drivers and the addition of the *sxt* driver itself, no changes are required to the kernel. Also, no changes are required to the shell: the user interface is provided entirely by *shl*, a program which is small and easily modifiable. Thus, those who do not use the facility do not suffer from its inclusion. A final advantage of this model is that no changes are required to existing software to run under the facility. Programs need never know whether they are talking to a real or a virtual terminal.

;login:

Loadable Virtual Disk Device Driver and Server

Thomas A. Alborough

Creare R&D Inc.

Box 71

Hanover, NH 03755

(603) 643-3800

A useful feature that can be provided by networking software is an ability for programs on one node to transparently access peripheral devices on a remote node. This can, for example, allow programs on a system with limited disk storage space to access additional space on another system. It can also allow access to devices like magnetic tape that are not available on the local system.

Creare has developed a multi-user, error-free communications link which connects a UNIX system to a remote computer. Use of this link requires a special protocol, so access is not transparent to existing programs. In order to permit transparent access to the remote system we have implemented a UNIX device driver for a virtual block device which presents to user processes all the characteristics of a real disk device. It can be read and written as a raw device or mounted as part of the file system.

OSx: Towards a Single UNIX System for Superminis

Ross Bott

Pyramid Technology Inc.

1295 Charleston Road

P.O. Box 7295

Mountain View, CA 94039

In the high-end minicomputer market, the available UNIX operating systems provide a difficult choice to the current and potential user. On one hand UNIX System V as provided by Western Electric provides a heavily-tested, relatively bug-free operating system with the promise of long-term support and continued software development. In addition, it provides a few features not available in other UNIX systems, and, because of its size, is likely to be the system of choice for many smaller minicomputers and microcomputers. On the other hand, the 4.2 BSD UNIX System released by University of California, Berkeley offers a variety of high-performance features necessary for most supermini applications, e.g., demand-paged virtual memory, a optimized file system, kernel-based networking, etc., plus many user interface features not available in System V. Whichever alternative users select, they must sacrifice performance and/or features.

This same quandary faces the implementor of a UNIX system on a supermini: The System V and 4.2 BSD system interfaces as well as the underlying code only partially overlap in some aspects and seriously conflict in others. Applications or utilities designed to run in one environment often have subtle incompatibilities when run in the other UNIX system.

This paper describes a project to design and implement a UNIX system which incorporates all of the 4.2 BSD performance features internally while providing a truly dual 4.2/System V interface to utilities and application programs: Neither interface is layered on the other, and users can transparently tailor their environment, both at the command level interface and for software development, to perform either like a fully-compatible 4.2BSD or a full System V interface, or, with some constraints, a hybrid interface. In addition, applications developers can work in a 4.2 environment while testing an application in a separate environment for System V compatibility, or vice versa.

;login:

Specific areas of incompatibility will be described in detail, both within the kernel, and at the command and application level. The solutions we used to resolve these conflicts will be laid out, along with some of the alternatives we chose not to take.

The implementation of OSx, the UNIX operating system which was the result of this project, is complete. The effect of the continued evolution of System V (and possibly 4.2) on OSx is considered, and some future directions are proposed.

New 1/2 Inch Tape Options and Trade-Offs for 4BSD UNIX on DEC VAX Processors

Robert J. Kridle

mt Xinu, Inc.
739 Allston Way
Berkeley, CA 94710
(415) 644-0146
ucbvax!mtxinu!kridle

Five tape units representing the spectrum of currently available streaming and start-stop tape equipment were benchmarked under 4.2 BSD UNIX on a DEC VAX 11/750. The performance of each unit was measured in typical UNIX tape applications such as dump/restor and tar archiving as well as under optimal circumstances for maximizing data transfer rates. The start-stop units tested include 45 and 125 ips 1600 bpi drives as well as a new, low cost 125 ips, 6250 bpi system. Two streaming tape units were evaluated. One streaming tape unit tested included a 64 Kbyte cache which allows it to successfully simulate a 125 ips start-stop unit. The other streaming unit, the DEC TU-80, features adaptive mode switching between two streaming speeds and a slow start-stop mode.

The performance of all units is reported and compared. It is shown that the new 4.2BSD "fast file system" makes differences in tape unit capabilities more important in file archiving applications. A suggestion is made for a modification to the 4BSD UNIX tape handler for the TU-80 which could improve its performance significantly.

;login:

Implementations II

8:30—10:00 Friday

A Layered Implementation of the UNIX Kernel on the HP9000 Series 500 Computer

Jeff Lindberg
Hewlett-Packard
3404 E Harmony Rd
Fort Collins, CO 80525
hplabs!hpfcla!jbl

An implementation of the UNIX operating system kernel has been layered on top of an existing operating system kernel for the HP9000 Series 500 computer. The mapping of UNIX functional requirements onto the capabilities of the underlying OS are presented in this paper, including the changes and extensions necessary to support UNIX semantic and performance requirements. The paper covers in retrospect the advantages and disadvantages of a layered approach.

Porting Xenix to the Unmapped 8086

John Bruno Hare
Dean Thomas
The Santa Cruz Operation
Santa Cruz, CA 95060

The Santa Cruz Operation is porting Xenix to several unmapped 8086-based computers. We have encountered and overcome several interesting problems in this process.

The 8086 microprocessor is one of the most popular inexpensive CPUs-on-a-chip to emerge in the personal computer market. The 8086, and its essentially equivalent sibling the 8088, combine the versatility of the 16-bit world with the simplicity of less sophisticated processors. Many 16-bit computers employ the 8086/8088, including the IBM PC, the Victor 9000, and Convergent Technologies' Integrated Work Station. These popular microcomputers, however, are all *unmapped* 8086's. This means they do not have a memory management unit (mmu) as most UNIX systems expect.

The Santa Cruz Operation has successfully implemented Xenix on a number of unmapped 8086-based microprocessors. As a result, the UNIX environment has migrated to a very popular low-end processor. This portends an immense widening of the market for Xenix and C based applications.

;login:

Writing Device Drivers For Xenix Systems

Jean McNamara
Paresh Vaish
Richard N. Bryant
Intel Corp.
5200 NE. Elam Young Pkwy.
Mail Stop: HF2-1-800
Hillsboro, Oregon 97123
decvax!tektronix!ogcvax!omsvax!rickb

Today most microcomputers and segments of the minicomputer industries are migrating toward non-proprietary operating systems. Commercial customers no longer want to be locked into one company's proprietary software products. Intel's systems group has adopted Xenix as its standard operating system for the commercial marketplace. Xenix is Microsoft Corp. direct port of AT&T UNIX with value added enhancements. UNIX has become the de facto standard for commercial microcomputers. The most important reason for this success is the Xenix/UNIX open system concept. Customers can purchase application packages from a variety of vendors. And, they can migrate to new hardware technologies while protecting their software investment. Peripheral and board manufacturers can add new controllers because Xenix provides a means to easily add new device drivers to the operating system.

The purpose of this paper is to discuss some of the major issues in writing a Xenix device driver. We will define Xenix device drivers then discuss the kernel facilities used by device drivers. Next a typical block device driver is discussed and its structure defined. A similar treatment is provided for a character device driver. A brief discussion of system configuration is given showing how device drivers are installed. Finally some driver programming issues are raised along with an example device driver.

Transparent Implementation of Shared Libraries

Curtis B. Downing
Bunker Ramo Information Systems
35 Nutmeg Drive
Trumbull Industrial Park
Trumbull, Connecticut 06609
203-386-2674
UUCP: [ucbvax!]decvax!bunker!curtis

Frank Farance
Systems Theory Design Corporation
555 Main Street, Suite 705
New York, New York 10044
212-355-4422
UUCP: [ucbvax!]decvax!std!ff

The authors have designed and implemented a shared library utility for software development. The shared library was implemented on a Motorola 68000-based UNIX V7 system. The main feature of this implementation is that the use of shared libraries is transparent to the programmer.

;login:

This paper includes a discussion of the following topics:

- * The initial motivation for shared libraries.
- * The transparency mechanism.
- * Compilation and loading of programs using shared libraries.
- * Dynamic linking of libraries to each other and to user programs.
- * Application tools for using libraries.
- * Development tools for building libraries.
- * Potential problems and solutions to updating and maintaining shared libraries.
- * Hardware and software requirements.
- * Kernel enhancements and requirements.
- * The potential use of shared libraries for other software systems (e.g. UNIX "libc" library).

UNIX File System Optimization on Microcomputer Systems

David Robboy

Intel Corp.
HF2-1-800

5200 N. E. Elam Young Parkway
Hillsboro, OR 97123
503-681-5490
omsvax!dgr

As microprocessors become more powerful, file systems become more of a bottleneck, particularly in low end microcomputer systems where the cost of disk devices must be minimized. This paper analyzes the sources of file system bottlenecks that arise in one UNIX environment, and presents some strategies for optimization.

We show that system throughput is dominated by the overhead of disk accesses more than by CPU processing or the data transfer rate, and is likely to become more so as processors and controllers improve. Thus reducing the number of disk reads, writes, and seeks is likely to improve system throughput proportionally more than reducing CPU activity or using a faster disk controller. To these ends, it is worthwhile to improve the buffer hit rate and to avoid the fragmentation of files.

Optimization methods are explored, including more intelligent buffer cacheing, track cacheing, larger block sizes, and periodic reorganization of the file system. The methods used in Berkeley UNIX are examined for their applicability to a microcomputer environment.

;login:

Communications

10:30—12:00 Friday

A UNIX to VAX/VMS Communications Link

Clifford N. Cary
Creare R&D Inc.
Box 71
Hanover, NH 03755
(603) 643-3800

Creare has implemented a multi-user, error-free communications link which connects a UNIX system to a VAX/VMS system. The link can be used simultaneously by many users for such purposes as file transfer, interactive terminal sessions, and direct access to remote devices. It uses an asynchronous serial line as the physical connection and is implemented entirely in the form of ordinary user programs. No operating system modifications are required on either system. The DDCMP data link protocol is used to detect and correct transmission errors. The UNIX code is presently implemented on a MASSCOMP MC-500 system.

On each system a pair of background processes run continuously to handle machine-to-machine data transfers and multiplexing and demultiplexing of messages from user processes. Processes which wish to use the link place requests in a queue, which is implemented as a named pipe. Users first issue an `Open_channel` request to become connected to a partner process on the remote system. The two processes then exchange messages across a private full-duplex channel until the session is terminated by a `Close_channel` request to the central managing process.

We have so far implemented three kinds of application programs which use the link: file transfer, interactive terminal session (similar to `cu`), and a virtual disk device driver and server which provides transparent access to the link by all UNIX software.

Loving UUCP, or How I Spent My Summer Vacation

P. Honeyman
D. A. Nowitz
Bell Laboratories
Murray Hill, New Jersey 07974

B. E. Redman
Bell Laboratories
Whippany, New Jersey 07981

In this talk, we describe an experimental version of UUCP, the standard UNIX to UNIX file transfer and remote execution facility.

Hacking UUCP is the rite of passage for most UNIX hackers; over the years, myriad niggling bugs, as well as several serious inherent design flaws and limitations, have been identified in existing versions of UUCP, prompting serious UNIX hackers (and some not so serious ones) to have hack at the source code. This has resulted in a multiplicity of versions, each with its own set of bizarre quirks, flavors, and bugs. (To be fair, though, we'll admit that many of the bugs are shared by the various versions.)

;login:

Over a six-week period, we attacked these problems, great and small, in what amounts to a total rewrite of the source code. Among our goals were to make reliable the dialing and login procedures, to ease the task of integrating new types of calling devices, to provide for enhanced protection in file transfer and remote execution (discussed in another talk), to improve the robustness of the spooling schemes, to develop a set of powerful administrative tools, to make the locking mechanism reliable and tolerant, to reduce the CPU overhead in file transfer, to provide a consistent version of UUCP for all combinations of UNIX versions and host processors, to enhance the remote execution capabilities, and, perhaps most important, to reduce the complexity of the algorithms used by UUCP and their implementations.

We accomplished most of our goals, e.g., if the calling equipment is not broken, and the remote is up, and at least one line in L.sys is accurate, then the connection goes through; we understand the vagaries of communicating through switches, such as Micom, Gandalf, Develcon, DATAKIT PS, and 3Com Ethernet, and a number of modems, e.g., VenTel, Vadec, Bell 212/801, and Hayes, as well as being able to connect through a switch to a modem; we reorganized the spool directory to maintain a separate directory for each site, thereby eliminating a particularly vexing instance of fatal positive feedback.

Although this burst of effort has dragged on into a seemingly endless process of searching for further nits to pick, this experimental version of UUCP is now running in a variety of environments, entailing a wide collection of combinations of processors, UNIX versions, and system loads, including several mail and USENET hubs. In this talk, we report on our history, goals, progress, and some amusing anecdotes describing our gaffes.

New Implementation of UUCP

D. A. Nowitz

AT&T Bell Laboratories

P. Honeyman

Princeton University

B. E. Redman

AT&T Bell Laboratories

The UUCP network software has been running for over 5 years. During that time, a number of bugs were uncovered and many enhancements were suggested. Early this year, a group of interested UUCP administrators met to discuss the production of a better version of UUCP. As a result, three people divided up the work and produced a version with the following major enhancements:

The security aspect was reworked to provide more restricted file access, to provide a clearer mechanism for specifying these permissions, and to provide easier initial setup.

A more flexible system for specifying and implementing various connection media is implemented; for example, Develcon and Micom switches, and Ventel dialers.

The spool directory is now implemented as multiple directories; this improves work search time and prevents blocking due to faulty communications to a particular system.

In addition, effort was applied to simplify the code, improve the algorithms, and provide additional administration tools. This talk will focus on the security aspects of the new software.

It is now possible to specify that some remote sites can only send files, they cannot request files; this provides a high degree of security while letting remotes communicate for purposes such as mail. In addition, another option can further restrict communications by not permitting the transfer of queued

;login:

local requests during a conversation initiated by a remote site. With this option, the local secure site must call the remote in order to transfer queued files; a masquerader can not call up and receive files destined for someone else.

The directories that can be accessed for reading or writing by a remote machine are specified by READ and WRITE options. Commands that are executed by "uuxqt" can be specified on a machine by machine basis. For machines that have special execution privileges, a VALIDATE option is available to check the login-id against the machine name.

All the options are specified in the PERMISSIONS file. The defaults for the file are for maximum protection; options must be specified to give greater freedom. An attempt was made to make it easier to read and understand this scheme; the entries in the file look like shell or makefile variables, for example LOGNAME=nuucp. In addition, a program is available that reads the PERMISSIONS file and prints an English version of how the UUCP programs will interpret the file.

Mapping the UUCP Network

Rob Kolstad

PARSEC SciCompCorp

Karen Summers-Horton

AT&T Bell Laboratories

The UUCP network encompasses those machine on the UNIX News Network (USENET) and more: approximately 1800 sites as of November 1, 1983. This talk details the difference between the UUCP network and USENET in addition to outlining the current problems in using the networks (notably those concerning reliable routing among sites: "how to get there from here"). If the UUCP network is to become an ARPA domain, reliable maps and site descriptions must be available to a "name server". This talk will explain our current plans for mapping not only the connectivity of the network but also the "quality" of the connections for use with routing programs such as Steve Bellovin's optimal path finder. The talk will include current schema for collecting, disseminating, and using the information.

;login:

Graphics

1:30—3:00 Friday

UNIX as a Development Base for High Performance Graphics Applications

Thomas Ferrin

Computer Graphics Laboratory

School of Pharmacy

University of California

San Francisco, CA 94143

UUCP address: ucbvax!ucsfcl!tef

MIDAS (Molecular Interactive Display and Simulation) is a large interactive molecular modeling graphics package developed on UNIX which offers features previously unavailable in macromolecular modeling software. Much of the success obtained with MIDAS can be attributed to the productive environment provided by UNIX. In particular, UNIX has provided an efficient high level language for writing both user programs and a complex device driver, character stream files with arbitrary byte positioning for developing a new data base organization and access primitives, access to system source code for implementing specialized performance enhancements, pipes for logical subprocess division, performance measuring tools such as *prof*, *gprof*, *vmstat* and *iostat*, program documentation aids, utilities for maintaining revisions to the source code, and, perhaps most importantly, an example of a basic and successful "tool building" philosophy.

This presentation describes the MIDAS molecular modeling system and the impact that the aforementioned UNIX facilities have had on its success. A 16mm color film illustrating some of the unique features available in MIDAS will be shown.

DAPS and GSDL: A Procedural Approach to Graphics

Christos Tountas

Graphic Information System Technology Inc.

Presentation graphic systems generally come in two varieties: subroutine packages and non-procedural end-user systems which are interactive and "user-friendly"

Subroutine packages, used with compiled languages like Fortran and C, are powerful but difficult to use since they require considerable system design and programming effort; a significant portion of application development responsibility is left to the user.

Non-procedural systems (whether menu- or language-driven) are inherently adequate only for the limited set of operations for which they were designed but become awkward when applied in a wider variety of situations. A first-time graphics user is usually enthusiastic about easy-to-use menu systems but quickly discovers that their friendliness is inversely proportional to their flexibility. Non-procedural systems, especially those that use menus, are not well suited for data-driven, high-volume applications or for unattended operation.

DAPS (Data Analysis and Presentation System) addresses these problems by offering a high-level, interpreted procedural language (which is easier to use than Fortran or C) as well as a library of menu-

;login:

and dialog-driven applications which are written in that language. Users who need to develop new applications may do so easily, often just by modifying or extending one of the library execs. Application execs are simple disk files which are dynamically loaded when needed, so large application systems may be easily built and modified with no need for global system re-linking with each modification. In addition, separate hierarchical menu systems may be merged with ease.

GSDL is a recent variant of DAPS suitable for architectural and engineering graphics. It utilizes the same basic procedural, parametrized language but contains a richer variety of graphic object creation and manipulation operations, graphic input, a two-level hierarchical 3D turtle geometry environment, and a variety of projection and hidden line elimination operations.

DAPS and GSDL may be loaded as a single system comprising the capabilities of both. The essential characteristic of both systems is that they integrate procedural and non-procedural, interactive operations in a way that gives the user great power for the development of specialized applications in addition to a growing library of standard ones.

The Design of a Dedicated Graphics Subsystem for a UNIX Machine or How to Make Pictures in Real Time

Jack Burness
MASSCOMP
1 Technology Park
Westford, MA 01886
(617) 692-6200
masscomp!jack

To many people a graphics system appears as a kind of nifty black box which users can instruct to display their data in all kinds of wondrous formats which dazzle the eye. The actual internal operations of the system and the thought processes which led to its design are often invisible to the user. This is the natural result of any complex system. However such knowledge is often interesting and useful, especially as more and more systems become graphically oriented. This presentation describes the thoughts that went into the design and implementation of one graphics system.

;login:

Image Processing Work Station

Dale K. Hensley

TRW

O2/1796

1 Space Park

Redondo Beach, CA 90278

(decvax!trw-unix!trwspf!hensley)

(ucbvax!trw-unix!trwspf!hensley)

(trwspf!hensley@lbl-csam)

This talk will discuss the work done at TRW in designing and implementing a portable set of software routines that perform image processing, graphic generation and signal analysis on a MICRO/PDP-11.

The design of the software has evolved over 2 years and can operate on a Comtal 8300, a Comtal Vision 1/20, and a Deanza IP8500. The software runs on a PDP-11/45, MICRO/PDP-11 and a VAX 750. This portion of the talk will focus on design decisions and portability pitfalls.

The workstation implementation of this software was done on the MICRO/PDP-11 for the Space Telescope Science Operations Ground System. This portion of the talk will focus on the current workstation configuration, the problems in moving software from a 32 bit machine (VAX) to a 16 bit machine (MICRO/PDP-11), the performance of the MICRO/PDP-11, and future workstations.

Real-Time UNIX

3:30—5:00 Friday

Life With UNIX in Real-Time

Steven T. Polya

Contel Information Systems

We have been bombarded with information about what can and cannot be done with UNIX in a real-time applications environment. Here we present a case study, one that addresses most of the important topics that arise when one wishes to adapt UNIX to such a real-time application. Specifically, we will address inter-process communication, a commonly addressable data base in RAM, priority scheduling problems, and the use of the UNIX kernel for special non-interruptable operations.

;login:

Real-Time Extensions to the UNIX Operating System

Bryon Look

Gary Ho

Data Systems Division
Hewlett Packard
11000 Wolfe Road
Cupertino, CA 95014

This paper discusses real time extensions to the UNIX operating system. The real-time requirements of technical computer systems are first discussed. The deficiencies of the UNIX operating system for real-time applications are then described. Finally, proposals for providing real-time extensions to UNIX are presented.

UNIX Performance Optimization UNIX/68000/SKYFFP

Joseph F. Germann

Sky Computers, Inc.

The combination of the UNIX operating system and the Motorola 68000 microprocessor has become a standard for the high-performance work station type computer system. With all of the attributes that both UNIX and the 68000 boast, one would think that there is little room left for significant performance improvement. The tight integration of the Sky Fast Floating Point (SKYFFP) processor into this computing environment will significantly increase system performance. This holds true not only for tasks that require floating point arithmetic, but also for those functions where time consuming integer arithmetic functions are performed.

Integrating a Peripheral Floating-Point Processor into UNIX

Eryk Vershen

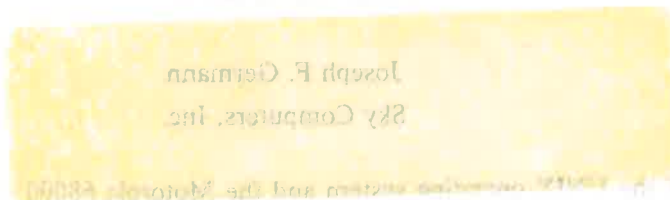
UniSoft Systems
739 Allston Way
Berkeley, CA 94710
ucbvax!unisoft!eryk
(415) 644-1230

The typical methods of supplying floating-point support are software simulation and tightly coupled hardware (e.g. a co-processor). A third technique is to have a peripheral floating-point processor.

This paper describes how such a processor (the Sky Computers Fast Floating-Point Processor) was integrated into UniSoft's UNIX-derived UniPlus+ kernels. The integration enables user access to the board with no extra per operation overhead. The attendant problems and tradeoffs with respect to efficiency and reliability are also discussed.

USENIX Association
P.O. Box 7
El Cerrito, CA 94530

First Class Mail



UniForum USENIX Program Begins on Page 5

UniForum USENIX Abstracts Begin on Page 9